

Ejercicio DyC

December 13, 2021

1 Inciso A

IDEA

Tengo n videos. Quiero combinar todos los consecutivos que sean del mismo tema. Puedo hacer lo siguiente.

Primero, recorro los n elementos, y voy agrupando en listas los que sé que voy a tener que combinar (por ejemplo: primeros 3 combinan, el 4 queda solo, otros 4 combinan). Después tengo que aplicar el algoritmo de combinación a cada caso que me quedó.

Cual es el peor caso? Aquel en el que todos los videos son del mismo tema, porque tengo que combinarlos a todos.

ALGORITMO

```

peliculas(in V: arreglo(video)) -> res: arreglo(video){
  lista videosAgrupados <- CrearLista()
  lista videosConsecutivosAgrupables <- CrearLista()
  videosConsecutivosAgrupables.agregar(V[0])
  for i in 1...tam(V)-1 {          // total: O(n) (cantidad de videos)
    if MismoTema(V[i-1], V[i]){    // O(1)
      videosConsecutivosAgrupables.agregar(V[i]) // O(1)
    } else {
      videosAgrupados.agregar(videosConsecutivosAgrupables) // O(1)
      videosConsecutivosAgrupables <- CrearLista() // O(1)
      // agrego una referencia al video en O(1)
      videosConsecutivosAgrupables.agregar(V[i])
    }
  }
  videosAgrupados.agregar(videosConsecutivosAgrupables) // O(1)
  res <- CrearArreglo(tam(videosAgrupados)) // O(g) con g = #grupos que pude armar

  i <- 0 // O(1)
  itVidAgrupados <- CrearIt(videosAgrupados) // O(1)
  while (HaySiguiete(itVidAgrupados)) {
    videosAgrupadosArr <- CrearArreglo(Siguiete(itVidAgrupados)) // O(l_j)
    // con l_j la longitud de la lista en la posición j de videosAgrupados
    res[i] <- combinarVideos(videosAgrupadosArr, 0, tam(videosAgrupadosArr)) // O(D.log(n))
    Avanzar(itVidAgrupados) // O(1)
    i <- i + 1 // O(1)
  }
}

```

Nota: combinarVideos usa el rango [desde, hasta). Llamo c a la cantidad de videos a unir: c = hasta-desde

```

combinarVideos(in V: arreglo(video), in desde: nat, in hasta: nat) -> res: video {
  if desde+1 == hasta {
    res <- V[desde] // O(1)
  } else {
    m <- hasta-desde/2 // O(1)
    vid0 <- combinarVideos(V, desde, m) // T(c/2)
    vid1 <- combinarVideos(V, m, hasta) // T(c/2)
    res <- combinar(vid0, vid1) // O(d(vid0)+d(vid1))
  }
}

```

COMPLEJIDAD Y CORRECTITUD

El algoritmo primero genera grupos con los videos que deben ser agrupados, priorizando el hecho de juntar la máxima cantidad de videos consecutivos del mismo tema. A estos conjuntos los mete en una lista de listas, donde cada una es un conjunto de videos a ser agrupados en orden. Luego, como ya tenemos divididos en subconjuntos los videos a combinar, sabemos que al combinar cada lista y agregar el resultado en el arreglo **res** estamos generando los videos de máxima longitud posible, con lo cual se satisface lo pedido en el enunciado.

La complejidad de la ejecución reside en dos partes. Primero, se recorren los n elementos y se mete cada uno de ellos en las listas distribuidas por tema (si $\text{MismoTema}(V1, V2)$, van a la misma lista. Si no, $V1$ va a la lista actual y para $V2$ creo una lista nueva). El costo de agregar en listas es constante y se agrega una referencia al video, por lo que no pago costo extra. Finalmente, paso por los n videos y agrego cada uno una única vez, lo que requiere una complejidad de $O(n)$.

Luego, pasar cada lista a un arreglo me requiere también, en sumatoria, $O(n)$, puesto que las listas totales tienen n videos, con lo cual esa será la cantidad exacta de memoria a utilizar. Esto lo expresé como la sumatoria de los l_j .

Finalmente, debo ver la complejidad de **combinarVideos**. El peor caso sería aquel en el que debo combinar todos los videos, dado que en cualquier otro caso realizaré menos combinaciones. Para esto, puedo armar un árbol de recursión, y dado que estamos partiendo el problema en 2 cada vez, vamos a tener un árbol binario completo de altura $h = \log(n)$. En ese caso, tendríamos lo siguiente:

- nivel h : Tiene 2^h videos que no requieren combinación.
- nivel $h-1$: Tiene $2^{(h-1)}$ combinaciones
- ...
- nivel 1: Tiene 1 combinacion

Ahora bien, en el nivel $h-1$, estaremos combinando $d(v1) + d(v2)$, $d(v3) + d(v4)$, y así. Esto significa que las complejidades pueden escribirse de este modo

- nivel $h-1$: $O((d(v1) + d(v2)) + (d(v3) + d(v4)) + \dots)$
- nivel $h-2$: $O((d(v1) + d(v2) + d(v3) + d(v4)) + \dots)$
- nivel 1: $O(d(v1) + d(v2) + d(v3) + d(v4) + \dots)$

Es decir que en el nivel $h-1$ se combinan $v1$ y $v2$, a complejidad $O(d(v1) + d(v2))$, y se combinan $v3$ y $v4$, a costo $O(d(v3) + d(v4))$. Pero en el siguiente nivel, al combinar esos dos videos, ejecutamos la mitad de combinaciones pero con el mismo costo temporal (porque se combinan dos videos que tienen mayor longitud) con lo cual la complejidad nivel a nivel se mantiene en $\sum_{i=1}^n d(v_i) = D$.

Por lo tanto, como el costo de cada nivel es D , y tenemos $\log(n)$ niveles, el costo de combinar los videos queda en $O(D \cdot \log(n))$.

De esta forma, sumando las complejidades, llegamos a $O(n + D \cdot \log(n))$